Windows 95/98 Device Driver for PCMCIA Notebook/Serial Highway Controller

BLN 2001-08 UM

March 2001

Eindhoven University of Technology
Department of Physics

Physical & Technical Laboratory Automation Group

Author: R. Smeets
Version: 1.0
Date: 16-03-2001

Table of contents

1. Introduction 3
1.1 General 3
1.2 Conventions used in this manual 4

2. Driver interface overview 5
2.1 Description of functional groups 5
2.2 Device driver error codes 6

3. Driver initialization and finalization functions 7

4. PhyBUS I/O and PhyBUS control functions 11

5. Exception handling 19
5.1 Description of exceptions 19
5.2 Exception functions 22

6. Miscellaneous functions 25

1. Introduction

1.1 General

This document describes the functions and the Application Programming Interface (API) of the Virtual Device Driver (VxD) for the Notebook/Serial Highway Controller (BLN 98-15) for Windows 95/98.

The driver offers a C interface to an application program.

The device driver, including the installation script, consists of the following files:

- NHC9815.VXD: this file contains the actual device driver for the PCMCIA card in the notebook
- NHCIO.DLL: this file interfaces the device driver to an application program.
- 9815.RBF, 9815B.RBF: these files contain the configuration for the programmable logic devices on the PCMCIA card.
- HIGHWAY.INF: this file contains the Windows 95/98 installation script for installing the PCMCIA card in a notebook or laptop computer

The files NHC9815.VXD, NHCIO.DLL, 9815.RBF and 9815B.RBF are installed in the \WINDOWS\SYSTEM directory.

An application program must include and link the following source files to obtain the entry points of the PhyBUS functions:

- NHC.H
- NHCLIB.C

A number of functions return the hardware status of the Notebook/Highway Controller or can modify its settings. See the user manual of the Notebook/Highway Controller for specific information (BLN 98-15 UM).

1.2 Conventions used in this manual

As the interface to the driver is intended to be used in a C programming environment with the Windows 95/98 operating system, the conventions used in this manual (with respect to types, include files, etc.) are Windows/C-like.

An application must always #include <windows.h> !

A short description of the types used by the driver:

WORD	word, 16 bits
DWORD	double word, 32 bits
VOID	if defined as a formal function parameter: function needs no actual parameter if defined as a function return value: function does not return a value
PWORD	pointer to a word
PDWORD	pointer to a double word
PVOID	if defined as a formal function parameter: actual parameter can be a pointer to any type if defined as a function return value: the return value can be a pointer to any type

2. Driver interface overview

2.1 Description of functional groups

The driver functions are divided into the following functional groups:

1. driver initialization and finalization functions.
2. PhyBUS I/O and PhyBUS control functions
3. PhyBUS exception functions (interrupts & bus errors)
4. miscellaneous

All driver functions start with the prefix “pb”. Every function returns a DWORD, indicating function success or failure: if the function return value is 0 (zero, PB_OK), the function was successful. A non-zero return value indicates failure. In this case, the error value identifies the specific failure. See section 2.2.

2.2 Device driver error codes

The error codes are defined in the file “nhc.h” and are given in table 2.1. Do not use literal error code values for processing errors, but use the symbolic names (table 2.1).

A number of error codes specify internal driver errors, and serve only as a means for tracking down internal system errors.

PB_OK	OK, no error
PB_ERR_NOT_OPEN	PhyBUS not open
PB_ERR_NO_IOPORT_EPLD	no IO port available for loading EPLD
PB_ERR_NO_IOPORT_PB	no IO port available for PhyBUS access
PB_ERR_RBF_OPEN	error opening RBF file
PB_ERR_RBF_READ	error reading RBF file
PB_ERR_RBF_CLOSE	error closing RBF file
PB_ERR_EPLD_CONFIG	EPLD configuration error
PB_ERR_EPLD_RESET	EPLD reset error
PB_ERR_CFG_NO_DEVNODE	error in locating device node
PB_ERR_CFG_BAD_DEVCONFIG	error in reading node configuration
PB_ERR_CFG_BAD_IO	error in IO config
PB_ERR_CFG_BAD_ATTMEM	error in attribute memory config
PB_ERR_CFG_BAD_IRQ	error in IRQ config
PB_ERR_INVALID_ADDRESS	invalid PhyBUS address
PB_ERR_PROTOCOL_SLAVE	protocol error slave side
PB_ERR_PROTOCOL_MASTER	protocol error master side
PB_ERR_BUSERROR_READ	PhyBUS read bus error
PB_ERR_BUSERROR_WRITE	PhyBUS write bus error
PB_ERR_BUSERROR_INIT	initPB bus error
PB_ERR_BAD_LINK	data errors testing Highway link
PB_ERR_MALLOC	general memory alloc error
PB_ERR_OPENKEY	error opening registry key
PB_ERR_CLOSEKEY	error closing registry key
PB_ERR_QUERYKEY	error query key
PB_ERR_ALLOCKEY	error memory alloc for key
PB_ERR_VXDFAILURE	driver failed
PB_ERR_NOVXD	NHC9815 VxD not found
PB_ERR_UNLOAD	VxD unload error
PB_ERR_CLOSEHANDLE	close handle failed
PB_ERR_THREADPRIORITY	error setting exception thread priority
PB_ERR_PROCESSPRIORITY	error setting current process priority
PB_ERR_NOTHREAD	no exception thread running

Table 2.1 Device driver error codes

3. Driver initialization and finalization functions

The following functions are used to initialize the driver and the interface to the driver. These functions must be called before using any PhyBUS function. Functions for finalizing the driver are also described.

Driver initialization and finalization functions should be used in the order as given in the following sample code:

```
DWORD result;

result = pbInitPhyBUSDLLEntries();
if (result != PB_OK) { /* error */
  exit(1);
  }

result = pbLoadVxD();
result = pbOpen();

/* run your application here */

result = pbClose();
result = pbUnloadVxD();
```

pbClose

Synopsis: #include “nhc.h”
DWORD pbClose(VOID);

Description: This function performs the (hardware) finalization of the Notebook/Serial Highway Controller and disables the controller for subsequent PhyBUS operations. Internal data structures are finalized, and internal system resources are freed.

Returns: The function returns PB_OK if finalization has succeeded. A non-zero return value indicates a failure.

See also: pbOpen

Example: -

---

pbInitPhyBUSDLLEntries

Synopsis: #include “nhc.h”
DWORD pbInitPhyBUSDLLEntries(VOID);

Description: This function initializes the driver entry points (addresses) of the PhyBUS functions in the file NHCLIB.C. This function must be called before calling any other driver function!

Returns: the function returns PB_OK if all driver entry points are initialized correctly. A non-zero return value specifies the number of non-initialized entries. An application program should not continue if the function returns non-zero!

See also: -

Example:

```
result = pbInitPhyBUSDLLEntries();
if (result != PB_OK) { /* error */
  exit(1);
  }
```

---

pbLoadVxD

Synopsis: #include “nhc.h”
DWORD pbLoadVxD(VOID);

Description: This function loads the Virtual Device Driver (VxD).

Returns: The function returns PB_OK if the driver is loaded succesfully. A non-zero return value indicates a load failure (e.g. the file NHC9815.VXD is not located in the \WINDOWS\SYSTEM directory).

See also: pbUnloadVxD

Example: -

---

pbOpen

Synopsis: #include “nhc.h”
DWORD pbOpen(VOID);

Description: This function performs the hardware initialization of the Notebook/Serial Highway Controller and enables the converter for subsequent PhyBUS operations. Internal data structures are initialized. I/O resources and system resources for handling interrupts are assigned.

If a PhyBUS crate with auto-configuration capabilities is used, this function also determines the current PhyBUS configuration. See section 4 for a description.

Returns: The function returns PB_OK if the Notebook/Serial Highway Controller is succesfully initialized. A non-zero return value indicates a failure.

See also: pbClose

Example: -

---

pbUnloadVxD

Synopsis: #include “nhc.h”
DWORD pbUnloadVxD(VOID);

Description: This function unloads the Virtual Device Driver, thereby freeing system resources. The driver can be unloaded by the application if the application program terminates, or if the driver is not needed anymore. If the application exits without calling pbUnloadVxD, the unloading is performed by the operating system.

Returns: The function returns PB_OK if the driver is unloaded succesfully. A non-zero return value indicates an unload failure.

See also: pbLoadVxD

Example: -

4. PhyBUS I/O and PhyBUS control functions

The PhyBUS I/O and PhyBUS control functions perform word or double word I/O operations on PhyBUS addresses.

Executing a read or a write access on a non-accessible PhyBUS address results in a bus error. As a default, a read bus error is signaled by the the return value PB_ERR_BUSERROR_READ of the invoked function, a write bus error is signaled by the return value PB_ERR_BUSERROR_WRITE of the invoked function. See section 6 for handling of PhyBUS bus errors.

---

pbClearBitsLong

Synopsis: #include “nhc.h”
DWORD pbClearBitsLong(WORD Address, DWORD Mask);

Description: This function clears the bits, specified in the doubleword Mask, at PhyBUS address Address. All other bits remain unchanged. The PhyBUS address must be a valid PhyBUS address in the range 0..4095 *and* the address must be even!

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbClearBitsWord, pbSetBitsWord, pbClearBitsLong, pbSetBitsLong

Example:

```
/* clear bits 2..0 at PhyBUS address 0x100 */
result = pbClearBitsLong(0x100, 0x7);

/* clear bit 31 at PhyBUS address 0x100 */
result = pbClearBitsLong(0x100, 0x80000000);
```

---

pbClearBitsWord

Synopsis: #include “nhc.h”
DWORD pbClearBitsWord(WORD Address, WORD Mask);

Description: This function clears the bits, specified in the word Mask, at PhyBUS address Address. All other bits remain unchanged. The PhyBUS address must be a valid PhyBUS address in the range 0..4095.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbSetBitsWord, pbClearBitsLong, pbSetBitsLong

Example:

```
/* clear bits 2..0 at PhyBUS address 0x37 */
result = pbClearBitsWord(0x37, 0x7);

/* clear bit 15 at PhyBUS address 0x37 */
result = pbClearBitsWord(0x37, 0x8000);
```

pbGetLong

Synopsis: #include “pb.h”
DWORD pbGetLong(WORD Address, PDWORD pData);

Description: This function reads a double word from the PhyBUS address Address. The double word read is stored in the location pointed to by pData. The PhyBUS address must be a valid PhyBUS address in the range 0..4095 *and* the address must be even!

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbGetWord, pbPutWord, pbPutLong

Example:

```
DWORD dataread;  /* receives result of pbGetLong */

/* read double word from PhyBUS address 0x012, and store it in
dataread */

result = pbGetLong(0x012, &dataread);
```

pbGetWord

Synopsis: #include “nhc.h”
DWORD pbGetWord(WORD Address, PWORD pData);

Description: This function reads a word from the PhyBUS address Address. The word read is stored in the location pointed to by pData. The PhyBUS address must be a valid PhyBUS address in the range 0..4095.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbGetWord, pbPutWord, pbGetLong, pbPutLong

Example:

```
WORD dataread;  /* receives result of pbGetWord */

/* read word from PhyBUS address 0x013, and store it in dataread */
result = pbGetWord(0x013, &dataread);
```

---

pbInit

Synopsis: #include “nhc.h”
DWORD pbInit(VOID);

Description: This function executes a global reset of ALL PhyBUS interfaces.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: -

Example: -

---

pbPutLong

Synopsis: #include “nhc.h”
DWORD pbPutLong(WORD Address, DWORD Data);

Description: This function writes the double word Data to the PhyBUS address Address. The PhyBUS address must be a valid PhyBUS address in the range 0..4094, AND the address must be even!

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbGetWord, pbPutWord, pbGetLong

Example:

```
/* write double word to PhyBUS address 0x12 */
result = pbPutLong(0x012, 0x12345678);
```

---

pbPutWord

Synopsis: #include “nhc.h”
DWORD pbPutWord(WORD Address, WORD Data);

Description: This function writes the word Data to the PhyBUS address Address. The PhyBUS address must be a valid PhyBUS address in the range 0..4095.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: -

Example:

```
/* write word to PhyBUS address 0x11 */
result = pbPutWord(0x011, 0x5678);
```

---

pbSetBitsLong

Synopsis: #include “nhc.h”
DWORD pbSetBitsLong(WORD Address, DWORD Mask);

Description: This function sets the bits, specified in the double word Mask, at PhyBUS address Address. All other bits remain unchanged. The PhyBUS address must be a valid PhyBUS address in the range 0..4094, AND the address must be even!

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbSetBitsWord, pbClearBitsWord, pbClearBitsLong

Example:

```
/* set bits 2..0 at PhyBUS address 0x40 */
result = pbSetBitsLong(0x40, 0x00000007);

/* set bit 15 at PhyBUS address 0x40 */
result = pbSetBitsLong(0x40, 0x8000);
```

pbSetBitsWord

Synopsis: #include “nhc.h”
DWORD pbSetBitsWord(WORD Address, WORD Mask);

Description: This function sets the bits, specified in the word Mask, at PhyBUS address Address. All other bits remain unchanged. The PhyBUS address must be a valid PhyBUS address in the range 0..4095.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: pbClearBitsWord, pbSetBitsLong, pbClearBitsLong

Example:

```
/* set bits 2..0 at PhyBUS address 0x37 */
result = pbSetBitsWord(0x37, 0x00000007);

/* set bit 15 at PhyBUS address 0x37 */
result = pbSetBitsWord(0x37, 0x8000);
```

5. Exception handling

5.1 Description of exceptions

The types of exceptions handled by the driver, are:

- PhyBUS interrupts. A PhyBUS interrupt can be generated by a PhyBUS interface which has its interrupts enabled.

- PhyBUS write bus errors. A PhyBUS write bus error occurs when writing to a non-accessible PhyBUS address. Possible causes are that no PhyBUS interface is present at the specified address, that the register at the specified address is read-only or that the specified address is (temporarily) unaccessible for write operations.

- PhyBUS read bus errors. A PhyBUS read bus error occurs when reading from a non-accessible PhyBUS address. Possible causes are that no PhyBUS interface is present at the specified address, that the register at the specified address is write-only or that the specified address is (temporarily) unaccessible for read operations.

- PhyBUS initialisation bus errors. A PhyBUS initialisation bus error occurs when issuing the pbInit() command. Possible causes include that the PhyBUS is not accessible because of an unpowered system crate.

- PhyBUS protocol errors. A PhyBUS protocol error occurs if the Serial Highway transmission protocol is violated. During normal operation this should *never* occur. However, for robustness the protocol exceptions can be signaled.

All exceptions are internally queued by the driver to be handled as Asynchronous Procedure Calls (APC). The driver runs a thread in which the default APC's for all exceptions are defined. By using the function pbSetExceptionHandler, the user can hook a user-defined exception handler to the default APC. The function pbSetExceptionHandler has the following prototype:

```
DWORD pbSetExceptionHandler( DWORD ExceptionType,
                             PEXCEPTIONHANDLER ExceptionHandler);
```

The parameter ExceptionType defines the type for which the specified exception handler must be installed, and must have one of the following values:

PhyBUS interrupts:	EXC_TYPE_PB_INTR
PhyBUS write bus errors:	EXC_TYPE_BER_WR
PhyBUS read bus errors:	EXC_TYPE_BER_RD
PhyBUS initialisation bus errors:	EXC_TYPE_PB_INIT
protocol errors:	EXC_TYPE_PER

The function pbSetExceptionHandler can only be used in an environment which is multithread-safe. Not all compiler vendors do provide these environments. If the environment is not multithread-safe, the application program may crash.

An exception handler is a pointer to a function, defined by the following typedef in NHC.H:

```
typedef VOID (*PEXCEPTIONHANDLER)(PAPCDATA);
```

PAPCDATA is a pointer to an APCDATA data structure:

```
typedef struct {
    DWORD intrtype;         /* interrupt type */
    DWORD intrcnt;          /* sequence number of interrupt */
    DWORD timehigh;         /* 64-bit time stamp of exception in units of 0.8 µs */
    DWORD timelow;
    DWORD intenq;           /* value of intenq register */
    DWORD lastaddress;      /* value of Last Address Register */
    DWORD lastdata;         /* value of Last Data Register */
} APCDATA,  *PAPCDATA;
```

The APCDATA data structure holds all the relevant information for handling any of the exceptions mentioned. The structure members have the following meaning:

- `intrtype`: this structure member contains the value representing the exception, and is the same as the parameter ExceptionType for which the corresponding exception handler has been installed. It can be used for verifying correct operation of the exception handler, or for dispatching another specific exception handler function.

- `intrcnt`: this structure member contains the current number of exceptions for this exception type.

- `timehigh, timelow`: these structure members contain the time stamp of the current exception. The 64-bit number represents the number of real time clock ticks that have elapsed since the current Windows session was started. Each clock tick represents a unit of 0.8 microseconds.

- `intenq`: this structure member contains only valid data if the exception is a PhyBUS interrupt (EXC_TYPE_PB_INTR). It contains a copy of the Interrupt Enquiry Register of the Notebook/Serial Highway Controller at the time of the PhyBUS interrupt. The Interrupt Enquiry Register is used to identify the PhyBUS device(s) which has (have) issued the interrupt(s).

- `lastaddress`: this structure member contains only valid data if the exception is a PhyBUS read bus error (EXC_TYPE_BER_RD) or a PhyBUS write bus error (EXC_TYPE_BER_WR). It contains the PhyBUS address where the bus error has occurred.

- `lastdata`: this structure member contains only valid data if the exception is a PhyBUS read bus error (EXC_TYPE_BER_RD) or a PhyBUS write bus error (EXC_TYPE_BER_WR). It contains the PhyBUS data where the bus error has occurred.

---

pbGetExceptionStatus

Synopsis: #include “nhc.h”
DWORD pbGetExceptionStatus(DWORD ExceptionType,
PAPCDATA pExceptionStatus);

Description: This function returns the current status of the specified exception type in an APCDATA structure. The parameter pExceptionStatus must hold the address of this APCDATA structure.

Calling this function copies the internal driver exception status to the user APCDATA structure and resets the *internal* exception counter for the specified exception type of the driver to zero.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: -

Example:

```
APCDATA apcbuf;      /* area for storing result */

/*get exception status for PhyBUS interrupts: */
result = pbGetExceptionStatus(EXC_TYPE_PB_INTR, &apcbuf);
```

---

pbSetExceptionHandler

Synopsis: #include “pb.h”
DWORD pbSetExceptionHandler(DWORD ExceptionType,
PEXCEPTIONHANDLER ExceptionHandler);

Description: This function installs a user-defined exception handler for the exception specified by ExceptionType. The function ExceptionHandler must be declared as:

VOID MyExceptionHandler(PAPCDATA pAPCData);

If the specified exception occurs, the function is invoked, and the parameter pAPCData holds the address of an internal driver APCDATA structure, of which the structure members can be read to obtain the data needed for processing the exception. The internal APCDATA structure is intended for read-only use.

The default (empty) exception handler can be set by calling pbSetExceptionHandler with argument NULL for the ExceptionHandler.

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: -

Example:

```
VOID MyPhyBUSInterruptHandler(PAPCDATA pAPCData)
{
printf(“Interrupt Enquiry Register = 0x%04x\n”, pAPCData->intenq);
}


/* set user-defined handler for PhyBUS interrupts */
result = pbSetExceptionHandler(EXC_TYPE_PB_INTR,
                MyPhyBUSInterruptHandler);


/*set default handler for PhyBUS interrupts */
result = pbSetExceptionHandler(EXC_TYPE_PB_INTR,NULL);
```

pbSetExceptionPriority

Synopsis:
```
#include "nhc.h"
DWORD pbSetExceptionPriority( DWORD ProcessPriority,
                              DWORD ThreadPriority);
```

Description: This function sets the process priority and the thread priority of *all* exception handlers. The parameter ProcessPriority is the priority of the current process from which the function is invoked. The parameter ThreadPriority is the priority of the thread in which the APC exception handler functions are running.

The following values for ProcessPriority are allowed (refer to the Windows SDK for more information on this subject):

THREAD_PRIORITY_ABOVE_NORMAL
THREAD_PRIORITY_BELOW_NORMAL
THREAD_PRIORITY_HIGHEST
THREAD_PRIORITY_IDLE
THREAD_PRIORITY_LOWEST
THREAD_PRIORITY_NORMAL
THREAD_PRIORITY_TIME_CRITICAL

The following values for ThreadPriority are allowed (refer to the Windows SDK for more information on this subject):

ABOVE_NORMAL_PRIORITY_CLASS
BELOW_NORMAL_PRIORITY_CLASS
HIGH_PRIORITY_CLASS
IDLE_PRIORITY_CLASS
NORMAL_PRIORITY_CLASS
REALTIME_PRIORITY_CLASS

Returns: The function returns PB_OK if it has succeeded. A non-zero return value indicates a failure.

See also: -

Example:
```
DWORD result;

/* heavy!! (locks out almost everything) */
result = pbSetExceptionPriority(REALTIME_PRIORITY_CLASS,
                                THREAD_PRIORITY_TIME_CRITICAL);
```

---

pbErrorMessage

Synopsis: #include “nhc.h”
PCHAR pbErrorMessage(DWORD MessageNumber);

Description: This function returns a pointer to a null-terminated string containing an error message corresponding with the value of MessageNumber.

Returns: pointer to a string containing the error message.

See also: -

Example:

```
DWORD result;

result = pbOpen();

/*show error, if any */
printf("%s\n", pbErrorMessage(result));
```

---

pbGetVersion

Synopsis: #include “nhc.h”
DWORD pbGetVersion(PDWORD pVersion);

Description: This function returns the version number of the DLL and the version number of the driver in the double word location pointed to by pVersion. After calling this function, the high word (bits 31..16) contains the DLL major and minor version number. The low word (bits 15..0) contains the VxD device driver major and minor version number.

Returns: The function returns PB_OK if the function call has succeeded. A non-zero return value indicates a failure.

See also: -

Example:

```
DWORD version; /* double word area */

/*get version */
result = pbGetVersion(&version);
printf("version = 0x%08x\n", version);
```